How to Calculate Feature Importance With Python

by **Jason Brownlee** on March 30, 2020 in **Data Preparation**

Last Updated on May 11, 2020

Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable.

There are many types and sources of feature importance scores, although popular examples include statistical correlation scores, coefficients calculated as part of linear models, decision trees, and permutation importance scores.

Feature importance scores play an important role in a predictive modeling project, including providing insight into the data, insight into the model, and the basis for dimensionality reduction and feature selection that can improve the efficiency and effectiveness of a predictive model on the problem.

In this tutorial, you will discover feature importance scores for machine learning in python

After completing this tutorial, you will know:

- The role of feature importance in a predictive modeling problem.
- How to calculate and review feature importance fr
- How to calculate and review permutation feature i

Let's get started.

- **Update May/2020**: Added example of feature sele

How to Calculate Feature Importance With Python
Photo by Bonnie Moreland, some rights reserved.

Tutorial Overview

This tutorial is divided into six parts; they are:

1. Feature Importance
2. Preparation
    1. Check Scikit-Learn Version
    2. Test Datasets
3. Coefficients as Feature Importance
    1. Linear Regression Feature Importance
    2. Logistic Regression Feature Importance
4. Decision Tree Feature Importance
    1. CART Feature Importance
    2. Random Forest Feature Importance
    3. XGBoost Feature Importance
5. Permutation Feature Importance
    1. Permutation Feature Importance for Regress
    2. Permutation Feature Importance for Classific
6. Feature Selection with Importance

Feature Importance

Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction.

Feature importance scores can be calculated for problems that involve predicting a numerical value, called regression, and those problems that involve predicting a class label, called classification.

The scores are useful and can be used in a range of situations in a predictive modeling problem, such as:

- Better understanding the data.
- Better understanding a model.
- Reducing the number of input features.

Feature importance scores can provide insight into the dataset. The relative scores can highlight which features may be most relevant to the target, and the converse, which features are the least relevant. This may be interpreted by a domain expert and could be used as the basis for gathering more or different data.

Feature importance scores can provide insight into the model. Most importance scores are calculated by a predictive model that has been fit on the dataset. Inspecting the importance score provides insight into that specific model and which features are the most important and least important to the model when making a prediction. This is a type of model interpretation that can be performed for those models that support it.

Feature importance can be used to improve a predictive model. This can be achieved by using the importance scores to select those features to delete (lowest scores) or those features to keep (highest scores). This is a type of feature selection and can simplify the problem that is being modeled, speed up the modeling process (deleting features is called dimensionality reduction), and in some cases, improve the performance of the model.

> *Often, we desire to quantify the strength of the relationship between the predictors and the outcome. […] Ranking predictors in this manner can be very useful when sifting through large amounts of data.*

— Page 463, Applied Predictive Modeling, 2013.

Feature importance scores can be fed to a wrapper m orm feature selection.

There are many ways to calculate feature importance s purpose.

Perhaps the simplest way is to calculate simple coeffic t variable. For more on this approach, see the tutorial:

- [How to Choose a Feature Selection Method for Machine Learning](https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/)

In this tutorial, we will look at three main types of more advanced feature importance; they are:

- Feature importance from model coefficients.
- Feature importance from decision trees.
- Feature importance from permutation testing.

Let's take a closer look at each.

Preparation

Before we dive in, let's confirm our environment and prepare some test datasets.

Check Scikit-Learn Version

First, confirm that you have a modern version of the scikit-learn library installed.

This is important because some of the models we will explore in this tutorial require a modern version of the library.

You can check the version of the library you have installed with the following code example:

```
# check scikit-learn version
import sklearn
print(sklearn.__version__)
```

Running the example will print the version of the library. At the time of writing, this is about version 0.22.

You need to be using this version of scikit-learn or higher.

```
0.22.1
```

Test Datasets

Next, let's define some test datasets that we can use as the basis for demonstrating and exploring feature importance scores.

Each test problem has five important and five unimpor ich
methods are consistent at finding or differentiating the

Classification Dataset

We will use the make_classification() function to creat

The dataset will have 1,000 examples, with 10 input fe
remaining five will be redundant. We will fix the rando les
each time the code is run.

An example of creating and summarizing the dataset is listed below.

```
# test classification dataset
from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, rando
# summarize the dataset
print(X.shape, y.shape)
```

Running the example creates the dataset and confirms the expected number of samples and features.

```
(1000, 10) (1000,)
```

Regression Dataset

We will use the make_regression() function to create a test regression dataset.

Like the classification dataset, the regression dataset will have 1,000 examples, with 10 input features, five of which will be informative and the remaining five that will be redundant.

```
# test regression dataset
from sklearn.datasets import make_regression
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# summarize the dataset
print(X.shape, y.shape)
```

Running the example creates the dataset and confirms the expected number of samples and features.

```
(1000, 10) (1000,)
```

Next, let's take a closer look at coefficients as importance scores.

Coefficients as Feature Importance

Linear machine learning algorithms fit a model where the prediction is the weighted sum of the input values.

Examples include linear regression, logistic regression, and extensions that add regularization, such as ridge regression and the elastic net.

All of these algorithms find a set of coefficients to use i n. These coefficients can be used directly as a crude typ

Let's take a closer look at using coefficients as feature will fit a model on the dataset to find the coefficients, then feature and finally create a bar chart to get an idea of

Linear Regression Feature Importance

We can fit a LinearRegression model on the regressio contains the coefficients found for each input variable.

These coefficients can provide the basis for a crude feature importance score. This assumes that the input variables have the same scale or have been scaled prior to fitting a model.

The complete example of linear regression coefficients for feature importance is listed below.

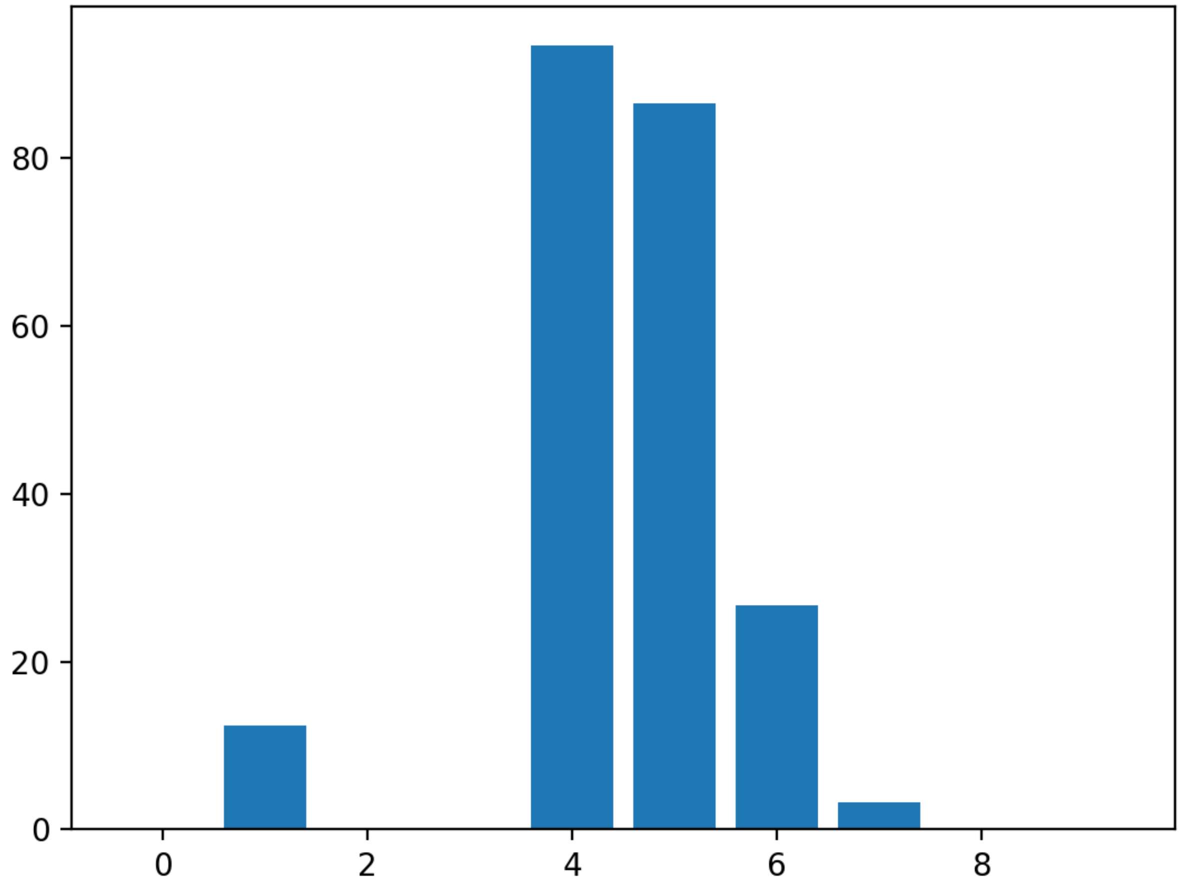
```
# linear regression feature importance
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = LinearRegression()
# fit the model
model.fit(X, y)
# get importance
importance = model.coef_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.

The scores suggest that the model found the five important features and marked all other features with a zero coefficient, essentially removing them from the model.

```
Feature: 0, Score: 0.00000
Feature: 1, Score: 12.44483
Feature: 2, Score: -0.00000
Feature: 3, Score: -0.00000
Feature: 4, Score: 93.32225
Feature: 5, Score: 86.50811
Feature: 6, Score: 26.74607
Feature: 7, Score: 3.28535
Feature: 8, Score: -0.00000
Feature: 9, Score: 0.00000
```

A bar chart is then created for the feature importance scores.



Bar Chart of Linear Regression Coefficients as Feature Importance Scores

This approach may also be used with Ridge and ElasticNet models.

Logistic Regression Feature Importance

We can fit a LogisticRegression model on the regression dataset and retrieve the *coeff_* property that contains the coefficients found for each input variable.

These coefficients can provide the basis for a crude fe... This assumes that the input variables have the same scale or have been scaled pr...

The complete example of logistic regression coefficien...

```
# logistic regression for feature importance
from sklearn.datasets import make_classificati
from sklearn.linear_model import LogisticRegre
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_f                                     ando
# define the model
model = LogisticRegression()
# fit the model
model.fit(X, y)
```

```
# get importance
importance = model.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

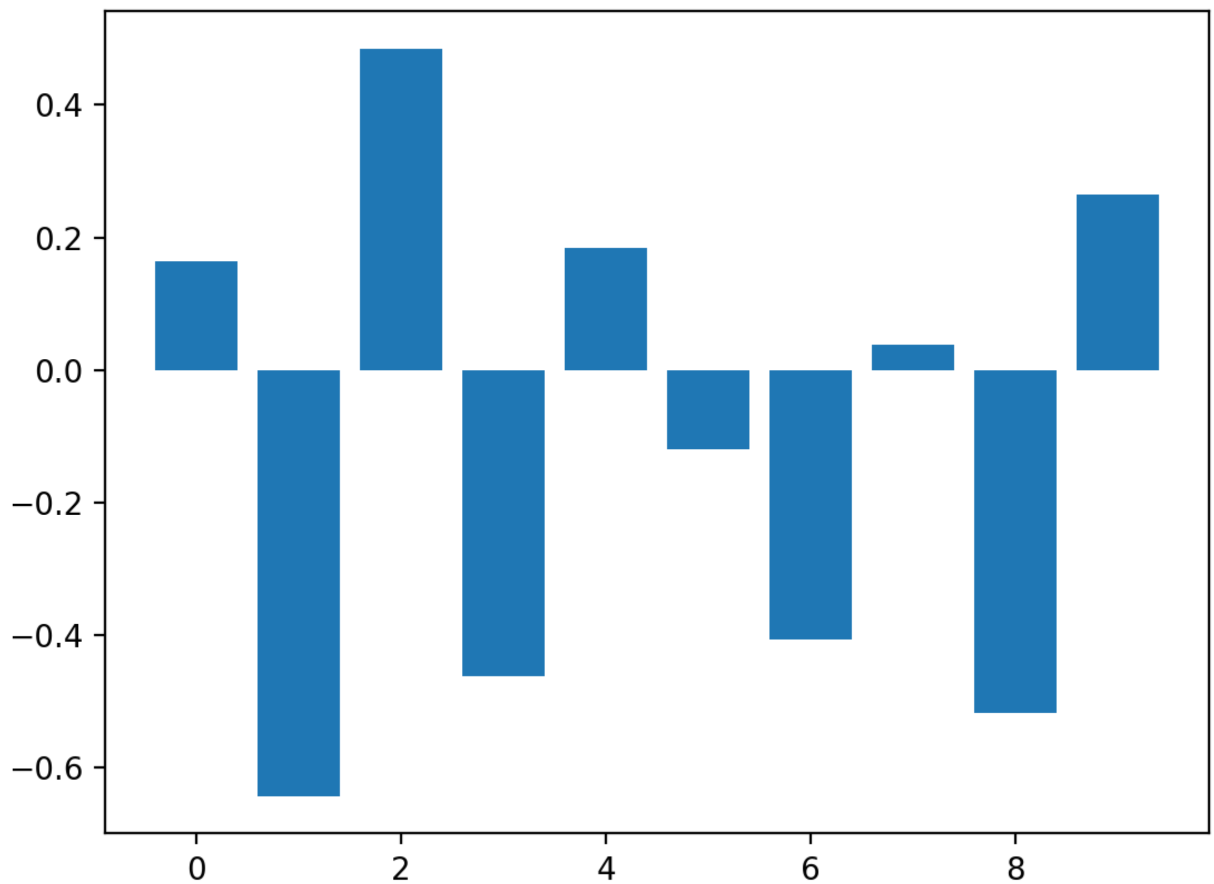
Running the example fits the model, then reports the coefficient value for each feature.

Recall this is a classification problem with classes 0 and 1. Notice that the coefficients are both positive and negative. The positive scores indicate a feature that predicts class 1, whereas the negative scores indicate a feature that predicts class 0.

No clear pattern of important and unimportant features can be identified from these results, at least from what I can tell.

```
Feature: 0, Score: 0.16320
Feature: 1, Score: -0.64301
Feature: 2, Score: 0.48497
Feature: 3, Score: -0.46190
Feature: 4, Score: 0.18432
Feature: 5, Score: -0.11978
Feature: 6, Score: -0.40602
Feature: 7, Score: 0.03772
Feature: 8, Score: -0.51785
Feature: 9, Score: 0.26540
```

A bar chart is then created for the feature importance scores.



Bar Chart of Logistic Regression Coefficients as Feature Importance Scores

Now that we have seen the use of coefficients as importance scores, let's look at the more common example of decision-tree-based importance scores.

Decision Tree Feature Importance

Decision tree algorithms like classification and regression trees (CART) offer importance scores based on the reduction in the criterion used to select split points, like Gini or entropy.

This same approach can be used for ensembles of de stochastic gradient boosting algorithms.

Let's take a look at a worked example of each.

CART Feature Importance

We can use the CART algorithm for feature importanc *DecisionTreeRegressor* and *DecisionTreeClassifier* cl

After being fit, the model provides a *feature_importances_* property that can be accessed to retrieve the relative importance scores for each input feature.

Let's take a look at an example of this for regression and classification.

CART Regression Feature Importance

The complete example of fitting a DecisionTreeRegressor and summarizing the calculated feature importance scores is listed below.

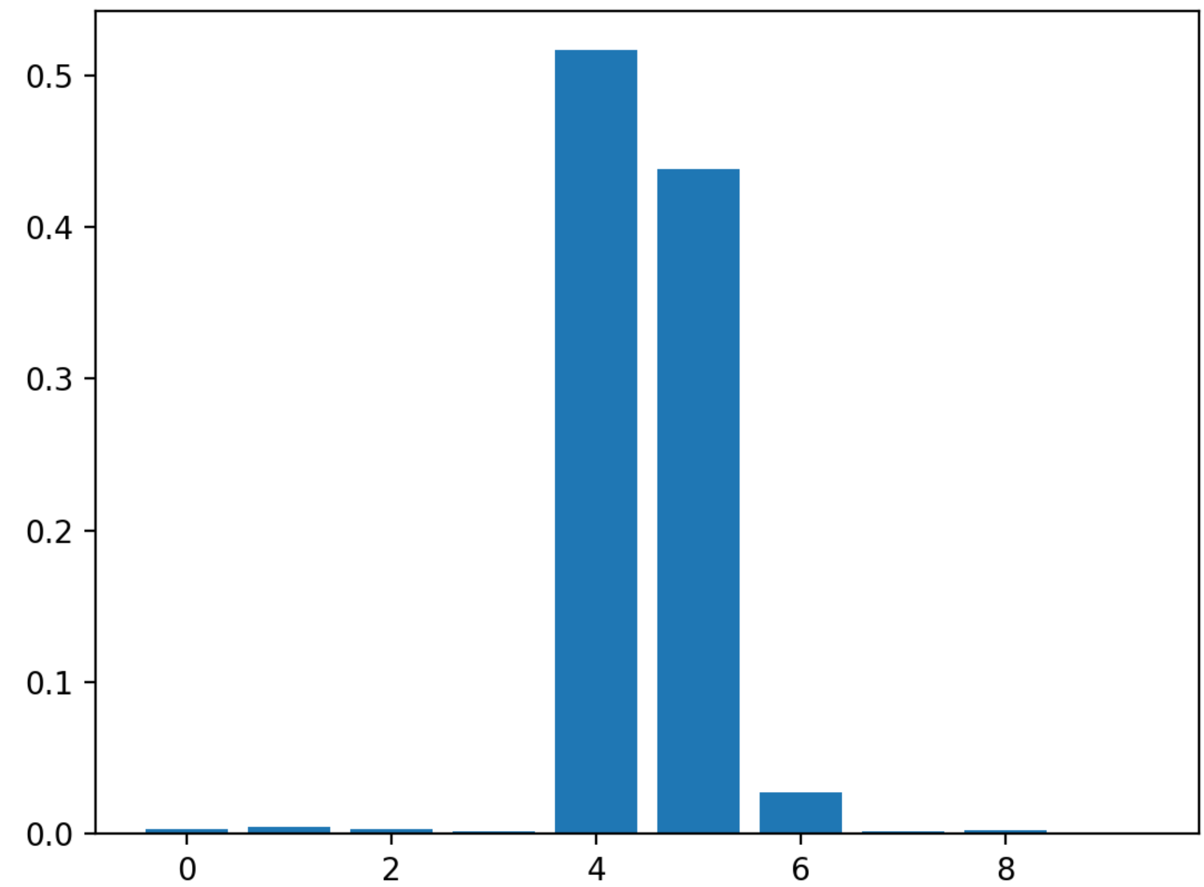
```
# decision tree for feature importance on a regression problem
from sklearn.datasets import make_regression
from sklearn.tree import DecisionTreeRegressor
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = DecisionTreeRegressor()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.

The results suggest perhaps three of the 10 features as being important to prediction.

```
Feature: 0, Score: 0.00294
Feature: 1, Score: 0.00502
Feature: 2, Score: 0.00318
Feature: 3, Score: 0.00151
Feature: 4, Score: 0.51648
Feature: 5, Score: 0.43814
Feature: 6, Score: 0.02723
Feature: 7, Score: 0.00200
Feature: 8, Score: 0.00244
Feature: 9, Score: 0.00106
```

A bar chart is then created for the feature importance scores.



Bar Chart of DecisionTreeRegressor Feature Importance Scores

CART Classification Feature Importance

The complete example of fitting a DecisionTreeClassifier and summarizing the calculated feature importance scores is listed below.

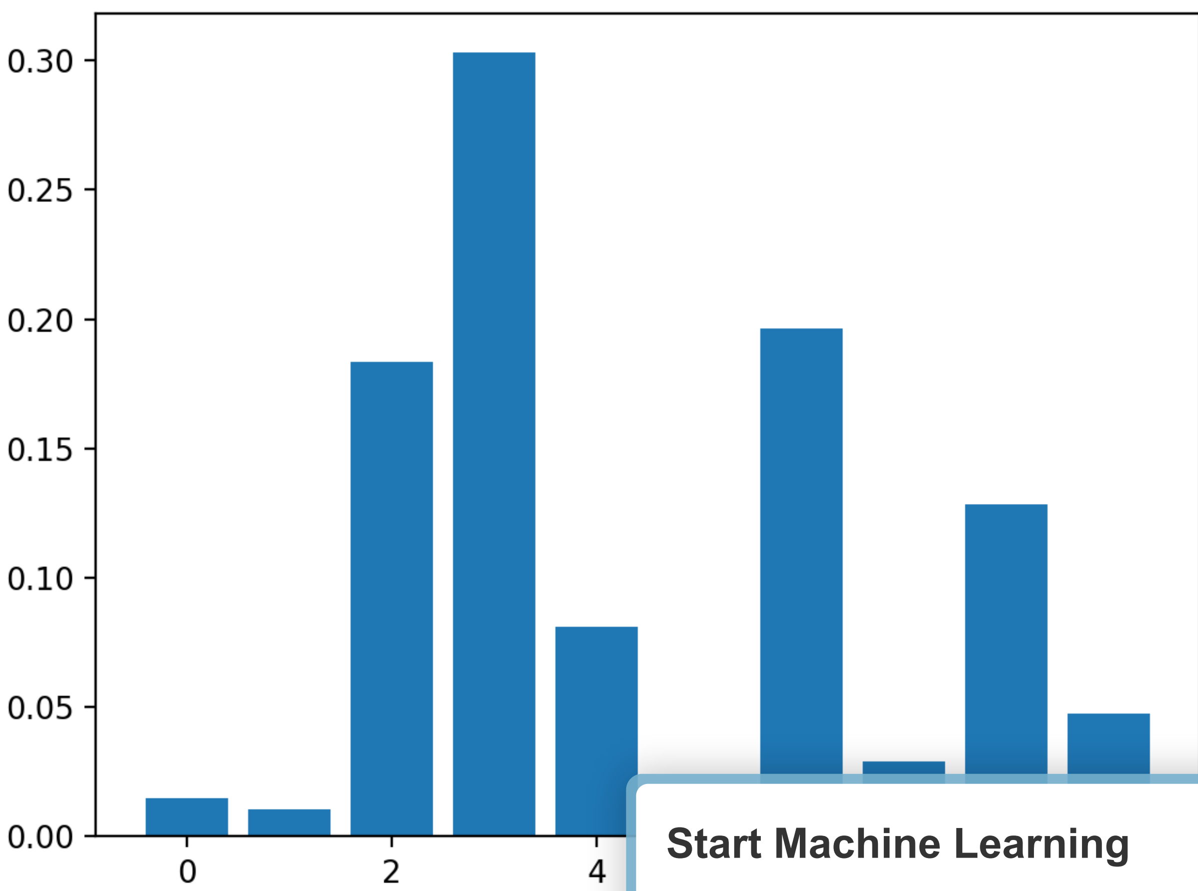
```
# decision tree for feature importance on a classification problem
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_                                              rando
# define the model
model = DecisionTreeClassifier()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))]
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.

The results suggest perhaps four of the 10 features as being important to prediction.

```
Feature: 0, Score: 0.01486
Feature: 1, Score: 0.01029
Feature: 2, Score: 0.18347
Feature: 3, Score: 0.30295
Feature: 4, Score: 0.08124
Feature: 5, Score: 0.00600
Feature: 6, Score: 0.19646
Feature: 7, Score: 0.02908
Feature: 8, Score: 0.12820
Feature: 9, Score: 0.04745
```

A bar chart is then created for the feature importance scores.



Bar Chart of DecisionTreeClass

Random Forest Feature Importance

We can use the Random Forest algorithm for feature i
RandomForestRegressor and *RandomForestClassifie*

After being fit, the model provides a *feature_importances_* property that can be accessed to retrieve the relative importance scores for each input feature.

This approach can also be used with the bagging and extra trees algorithms.

Let's take a look at an example of this for regression and classification.

Random Forest Regression Feature Importance

The complete example of fitting a RandomForestRegressor and summarizing the calculated feature importance scores is listed below.

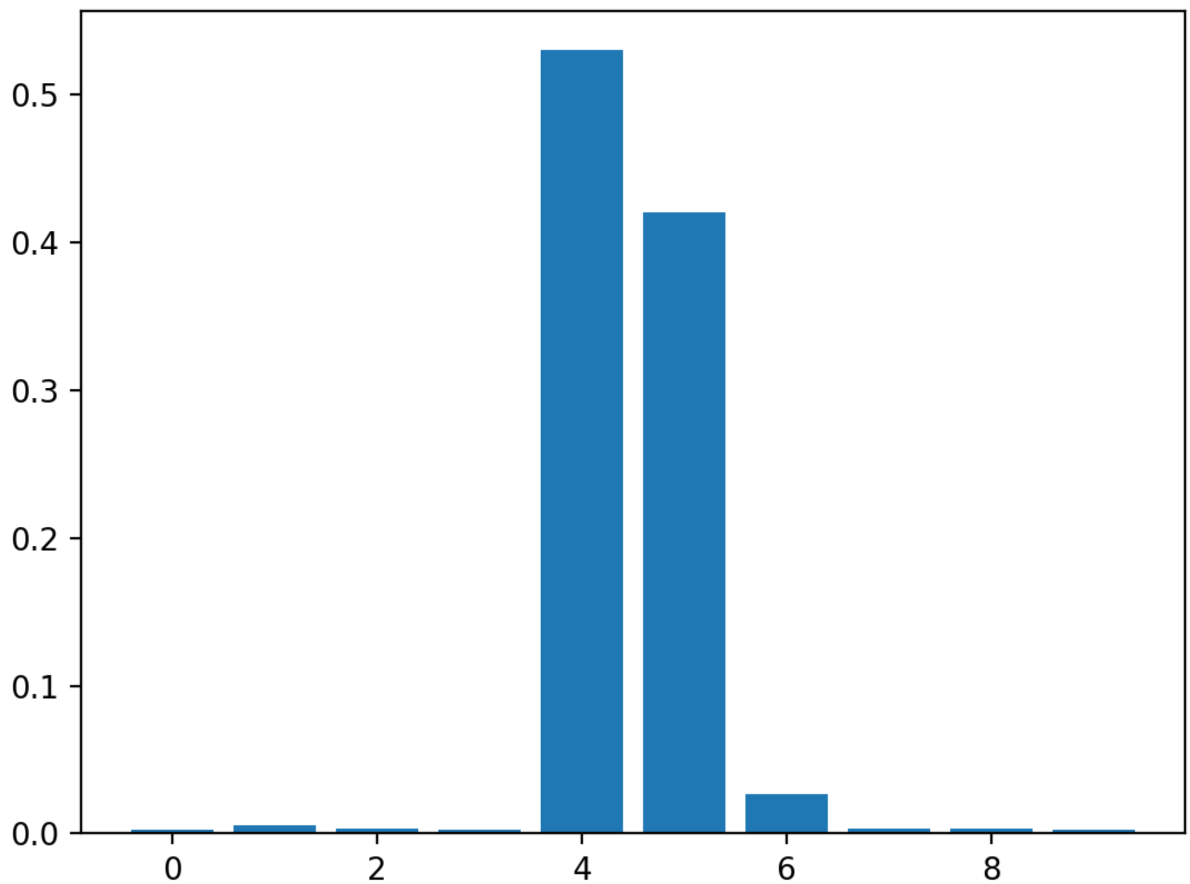
```
# random forest for feature importance on a regression problem
from sklearn.datasets import make_regression
from sklearn.ensemble import RandomForestRegressor
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = RandomForestRegressor()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.

The results suggest perhaps two or three of the 10 features as being important to prediction.

```
Feature: 0, Score: 0.00280
Feature: 1, Score: 0.00545
Feature: 2, Score: 0.00294
Feature: 3, Score: 0.00289
Feature: 4, Score: 0.52992
Feature: 5, Score: 0.42046
Feature: 6, Score: 0.02663
Feature: 7, Score: 0.00304
Feature: 8, Score: 0.00304
Feature: 9, Score: 0.00283
```

A bar chart is then created for the feature importance s



Bar Chart of RandomForestRegressor Feature Importance Scores

Random Forest Classification Feature Importance

The complete example of fitting a RandomForestClassifier and summarizing the calculated feature importance scores is listed below.

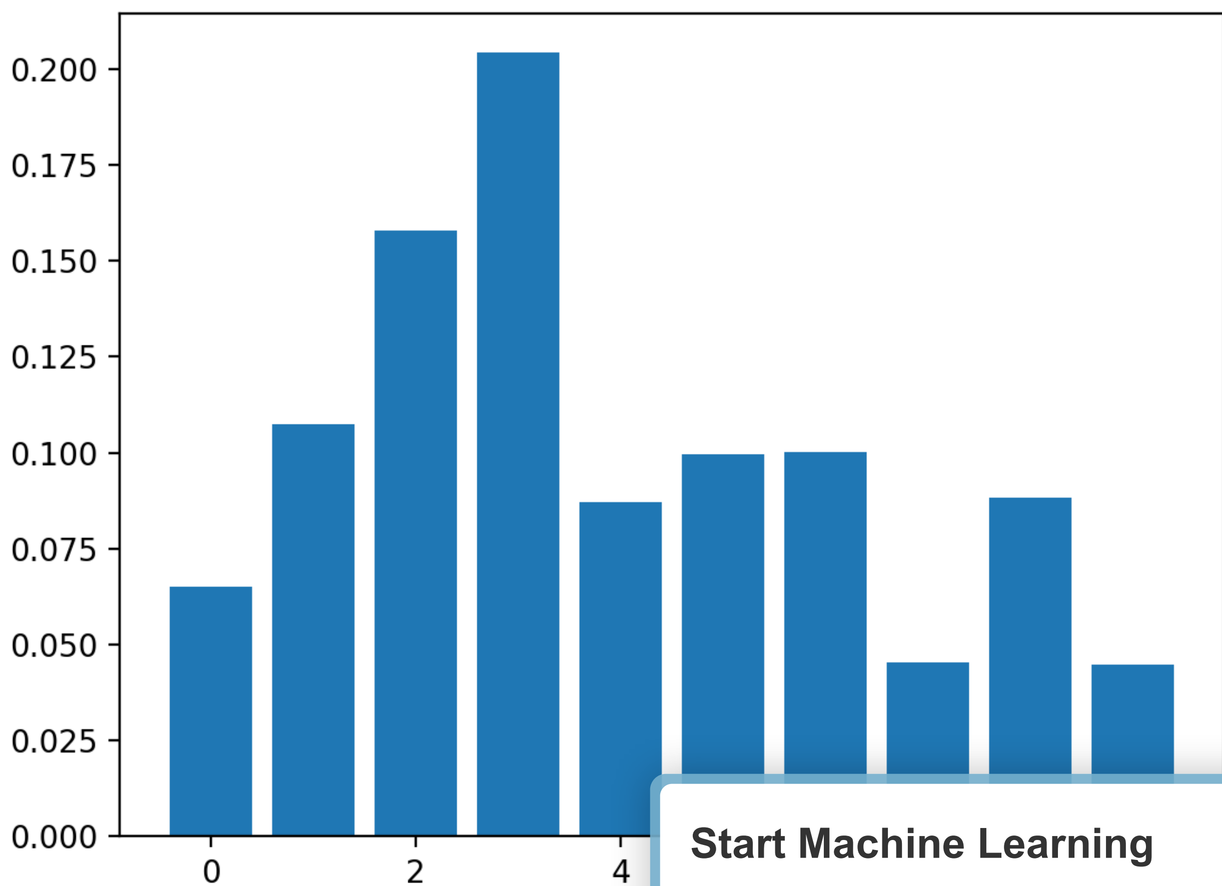
```
# random forest for feature importance on a classification problem
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_f                                                    rand
# define the model
model = RandomForestClassifier()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))]
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.

The results suggest perhaps two or three of the 10 features as being important to prediction.

```
Feature: 0, Score: 0.06523
Feature: 1, Score: 0.10737
Feature: 2, Score: 0.15779
Feature: 3, Score: 0.20422
Feature: 4, Score: 0.08709
Feature: 5, Score: 0.09948
Feature: 6, Score: 0.10009
Feature: 7, Score: 0.04551
Feature: 8, Score: 0.08830
Feature: 9, Score: 0.04493
```

A bar chart is then created for the feature importance scores.



Bar Chart of RandomForestClas

XGBoost Feature Importance

XGBoost is a library that provides an efficient and effe
boosting algorithm.

This algorithm can be used with scikit-learn via the *XGBRegressor* and *XGBClassifier* classes.

After being fit, the model provides a *feature_importances_* property that can be accessed to retrieve the relative importance scores for each input feature.

This algorithm is also provided via scikit-learn via the *GradientBoostingClassifier* and *GradientBoostingRegressor* classes and the same approach to feature selection can be used.

First, install the XGBoost library, such as with pip:

```
sudo pip install xgboost
```

Then confirm that the library was installed correctly and works by checking the version number.

```
# check xgboost version
import xgboost
print(xgboost.__version__)
```

Running the example, you should see the following version number or higher.

```
0.90
```

For more on the XGBoost library, start here:

- XGBoost with Python

Let's take a look at an example of XGBoost for feature importance on regression and classification problems.

XGBoost Regression Feature Importance

The complete example of fitting a XGBRegressor and summarizing the calculated feature importance scores is listed below.

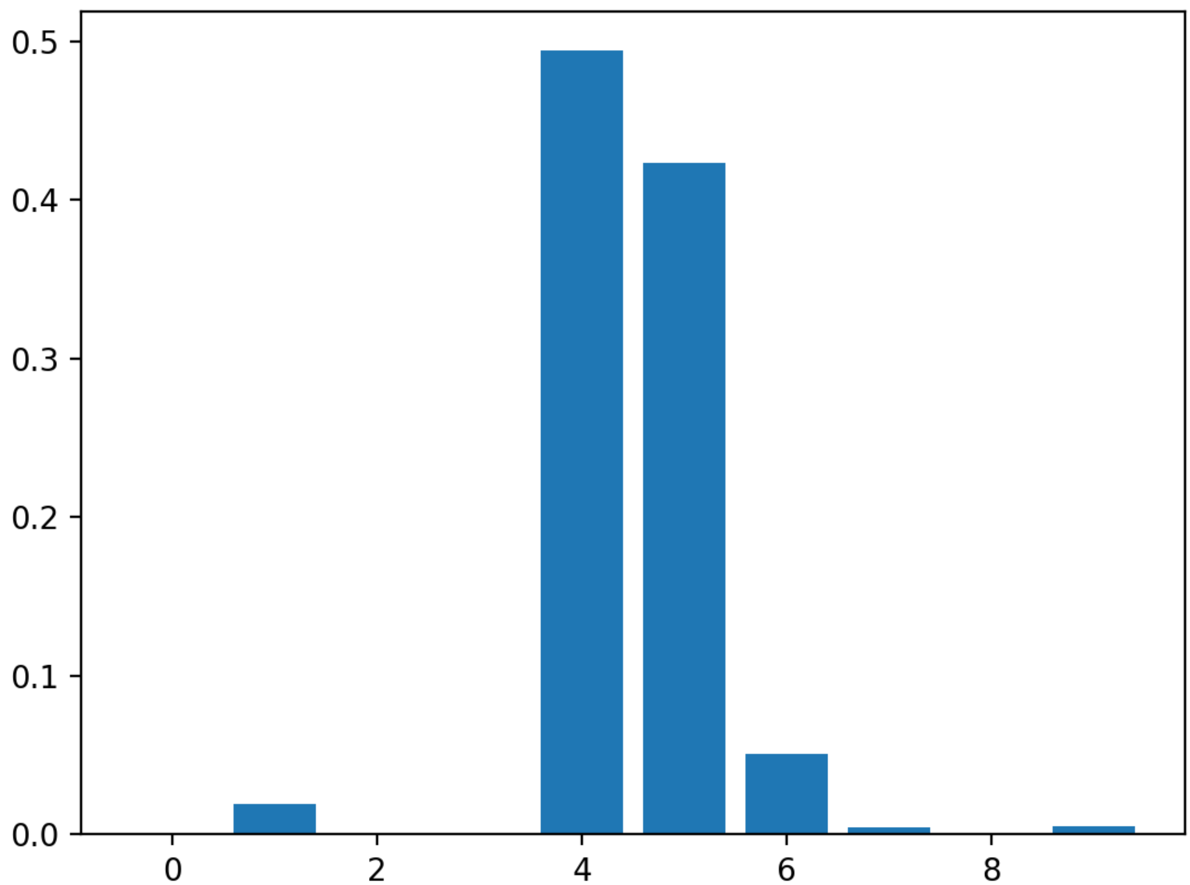
```
# xgboost for feature importance on a regression problem
from sklearn.datasets import make_regression
from xgboost import XGBRegressor
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = XGBRegressor()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))]
pyplot.show()
```


Running the example fits the model, then reports the c

The results suggest perhaps two or three of the 10 fea

```
Feature: 0, Score: 0.00060
Feature: 1, Score: 0.01917
Feature: 2, Score: 0.00091
Feature: 3, Score: 0.00118
Feature: 4, Score: 0.49380
Feature: 5, Score: 0.42342
Feature: 6, Score: 0.05057
Feature: 7, Score: 0.00419
Feature: 8, Score: 0.00124
Feature: 9, Score: 0.00491
```

A bar chart is then created for the feature importance scores.



Bar Chart of XGBRegressor

XGBoost Classification Feature Importance

The complete example of fitting an XGBClassifier and
scores is listed below.

```
# xgboost for feature importance on a classif
from sklearn.datasets import make_classificat
from xgboost import XGBClassifier
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_                                        ando
```

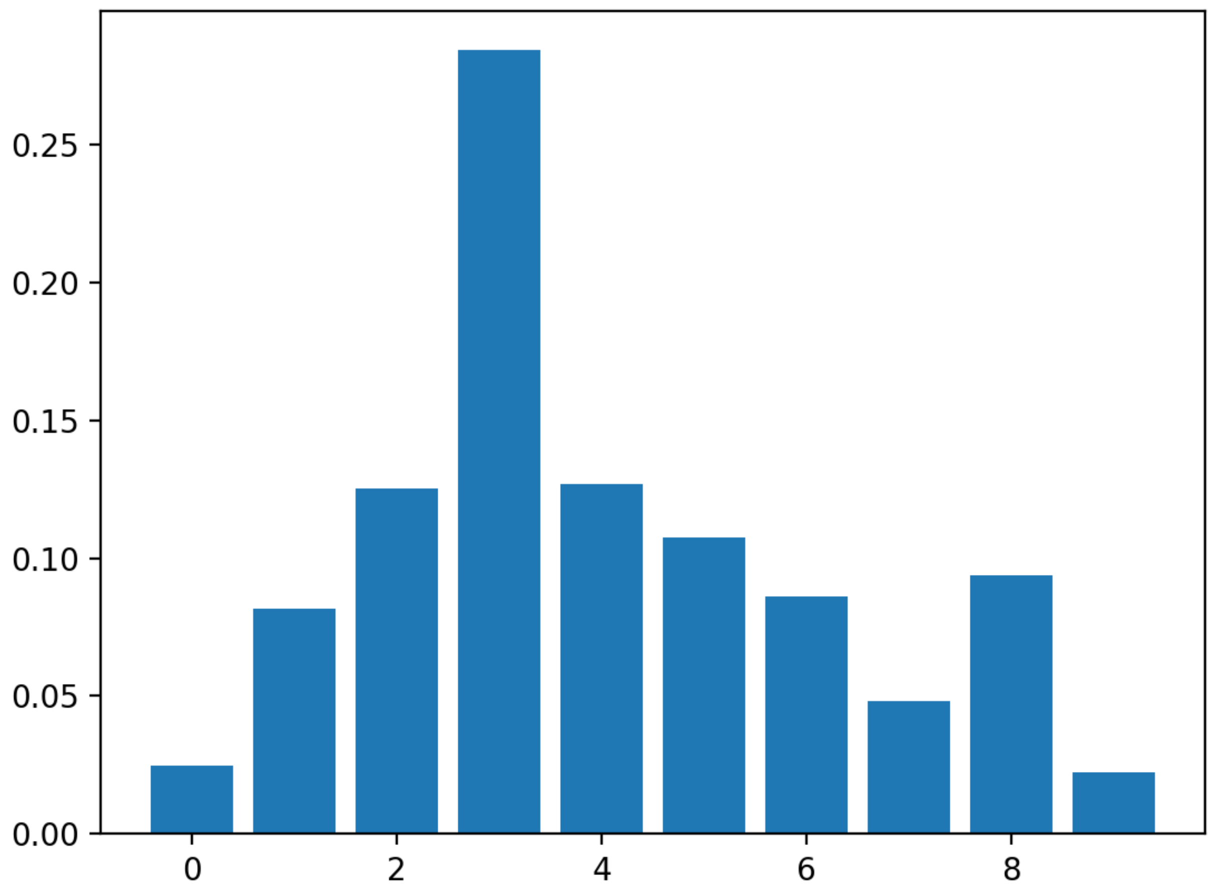
```
# define the model
model = XGBClassifier()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model then reports the coefficient value for each feature.

The results suggest perhaps seven of the 10 features as being important to prediction.

```
Feature: 0, Score: 0.02464
Feature: 1, Score: 0.08153
Feature: 2, Score: 0.12516
Feature: 3, Score: 0.28400
Feature: 4, Score: 0.12694
Feature: 5, Score: 0.10752
Feature: 6, Score: 0.08624
Feature: 7, Score: 0.04820
Feature: 8, Score: 0.09357
Feature: 9, Score: 0.02220
```

A bar chart is then created for the feature importance scores.



Bar Chart of XGBClassifier Feature Importance Scores

Permutation Feature Importance

Permutation feature importance is a technique for calculating relative importance scores that is independent of the model used.

First, a model is fit on the dataset, such as a model that does not support native feature importance scores. Then the model is used to make predictions on a dataset, although the values of a feature (column) in the dataset are scrambled. This is repeated for each featu
repeated 3, 5, 10 or more times. The result is a mean i
distribution of scores given the repeats).

This approach can be used for regression or classifica
chosen as the basis of the importance score, such as t acy
for classification.

Permutation feature selection can be used via the per del,
a dataset (train or test dataset is fine), and a scoring fu

Let's take a look at this approach to feature selection with an algorithm that does not support feature selection natively, specifically k-nearest neighbors.

Permutation Feature Importance for Regression

The complete example of fitting a KNeighborsRegressor and summarizing the calculated permutation feature importance scores is listed below.

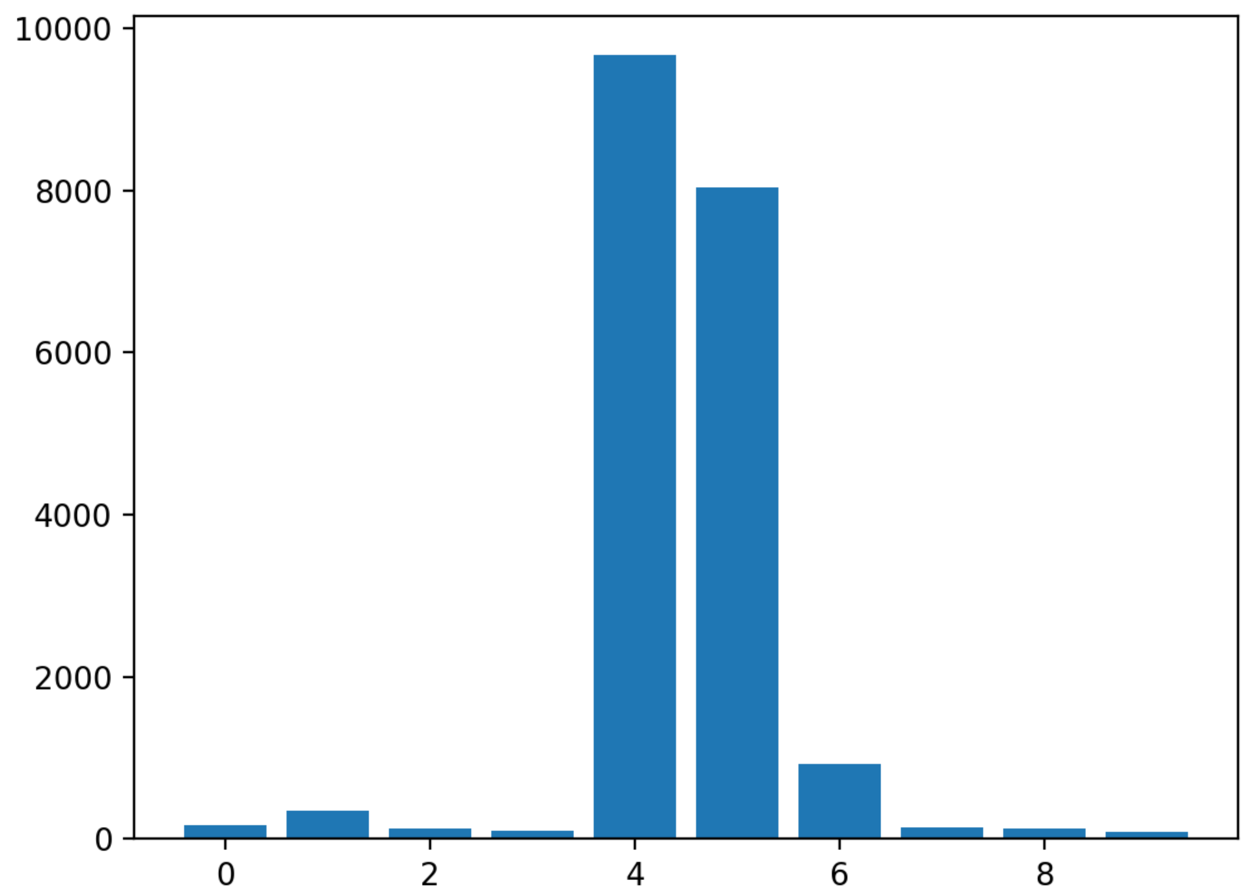
```
# permutation feature importance with knn for regression
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.inspection import permutation_importance
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = KNeighborsRegressor()
# fit the model
model.fit(X, y)
# perform permutation importance
results = permutation_importance(model, X, y, scoring='neg_mean_squared_error')
# get importance
importance = results.importances_mean
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.

The results suggest perhaps two or three of the 10 features as being important to prediction.

```
Feature: 0, Score: 175.52007
Feature: 1, Score: 345.80170
Feature: 2, Score: 126.60578
Feature: 3, Score: 95.90081
Feature: 4, Score: 9666.16446
Feature: 5, Score: 8036.79033
Feature: 6, Score: 929.58517
Feature: 7, Score: 139.67416
Feature: 8, Score: 132.06246
Feature: 9, Score: 84.94768
```

A bar chart is then created for the feature importance



Bar Chart of KNeighborsRegressor With Permutation Feature Importance Scores

Permutation Feature Importance for Classification

The complete example of fitting a KNeighborsClassifier and summarizing the calculated permutation feature importance scores is listed below.

```
# permutation feature importance with knn for classification
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.inspection import permutation_im
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_                                    rando
# define the model
model = KNeighborsClassifier()
# fit the model
model.fit(X, y)
# perform permutation importance
results = permutation_importance(model, X, y,
# get importance
importance = results.importances_mean
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
```

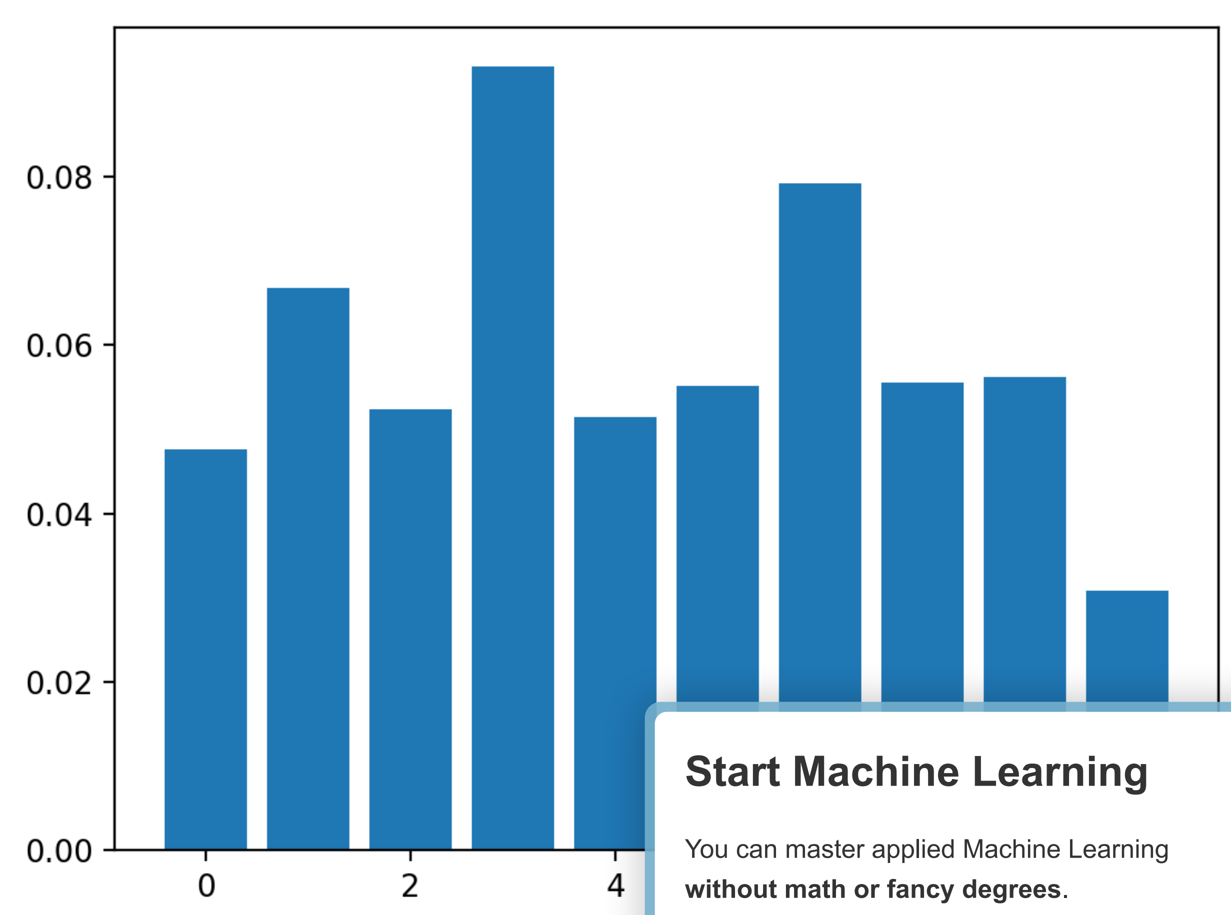
```
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.

The results suggest perhaps two or three of the 10 features as being important to prediction.

```
Feature: 0, Score: 0.04760
Feature: 1, Score: 0.06680
Feature: 2, Score: 0.05240
Feature: 3, Score: 0.09300
Feature: 4, Score: 0.05140
Feature: 5, Score: 0.05520
Feature: 6, Score: 0.07920
Feature: 7, Score: 0.05560
Feature: 8, Score: 0.05620
Feature: 9, Score: 0.03080
```

A bar chart is then created for the feature importance scores.



Bar Chart of KNeighborsClassifier With F

Feature Selection with Importar

Feature importance scores can be used to help interpret the data, but they can also be used directly to help rank and select features that are most useful to a predictive model.

We can demonstrate this with a small example.

Recall, our synthetic dataset has 1,000 examples each with 10 input variables, five of which are redundant and five of which are important to the outcome. We can use feature importance scores to help select the five variables that are relevant and only use them as inputs to a predictive model.

First, we can split the training dataset into train and test sets and train a model on the training dataset, make predictions on the test set and evaluate the result using classification accuracy. We will use a logistic regression model as the predictive model.

This provides a baseline for comparison when we remove some features using feature importance scores.

The complete example of evaluating a logistic regression model using all features as input on our synthetic dataset is listed below.

```
# evaluation of a model using all features
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# define the dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, rando
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# fit the model
model = LogisticRegression(solver='liblinear')
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
accuracy = accuracy_score(y_test, yhat)
print('Accuracy: %.2f' % (accuracy*100))
```

Running the example first the logistic regression model on the training dataset and evaluates it on the test set.

Your specific results may vary given the stochastic nature of the learning algorithm. Try running the example a few times.

In this case we can see that the model achieved the c g all features in the dataset.

```
Accuracy: 84.55
```

Given that we created the dataset, we would expect b nput variables.

We could use any of the feature importance scores ex ure importance scores provided by random forest.

We can use the SelectFromModel class to define both the model we wish to calculate importance scores, RandomForestClassifier in this case, and the number of features to select, 5 in this case.

```
...
# configure to select a subset of features
fs = SelectFromModel(RandomForestClassifier(n_estimators=200), max_features=5)
```

We can fit the feature selection method on the training dataset.

This will calculate the importance scores that can be used to rank all input features. We can then apply the method as a transform to select a subset of 5 most important features from the dataset. This transform will be applied to the training dataset and the test set.

```
...
# learn relationship from training data
fs.fit(X_train, y_train)
# transform train input data
X_train_fs = fs.transform(X_train)
# transform test input data
X_test_fs = fs.transform(X_test)
```

Tying this all together, the complete example of using random forest feature importance for feature selection is listed below.

```
# evaluation of a model using 5 features chosen with random forest importance
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# feature selection
def select_features(X_train, y_train, X_test):
    # configure to select a subset of features
    fs = SelectFromModel(RandomForestClassifier(n_estimators=1000), max_features=5)
    # learn relationship from training data
    fs.fit(X_train, y_train)
    # transform train input data
    X_train_fs = fs.transform(X_train)
    # transform test input data
    X_test_fs = fs.transform(X_test)
    return X_train_fs, X_test_fs, fs

# define the dataset
X, y = make_classification(n_samples=1000, n_
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_
# feature selection
X_train_fs, X_test_fs, fs = select_features(X_
# fit the model
model = LogisticRegression(solver='liblinear')
model.fit(X_train_fs, y_train)
# evaluate the model
yhat = model.predict(X_test_fs)
# evaluate predictions
accuracy = accuracy_score(y_test, yhat)
print('Accuracy: %.2f' % (accuracy*100))
```

Running the example first performs feature selection on the dataset, then fits and evaluates the logistic regression model as before.

Your specific results may vary given the stochastic nature of the learning algorithm. Try running the example a few times.

In this case, we can see that the model achieves the same performance on the dataset, although with half the number of input features. As expected, the feature importance scores calculated by random forest allowed us to accurately rank the input features and delete those that were not relevant to the target variable.

```
Accuracy: 84.55
```

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Related Tutorials

- How to Choose a Feature Selection Method For Machine Learning
- How to Perform Feature Selection with Categorical Data
- Feature Importance and Feature Selection With XGBoost in Python
- Feature Selection For Machine Learning in Python
- An Introduction to Feature Selection

Books

- Applied Predictive Modeling, 2013.

APIs

- Feature selection, scikit-learn API.
- Permutation feature importance, scikit-learn API.
- sklearn.datasets.make_classification API.
- sklearn.datasets.make_regression API.
- XGBoost Python API Reference.
- sklearn.inspection.permutation_importance API.

Summary

In this tutorial, you discovered feature importance sco

Specifically, you learned:

- The role of feature importance in a predictive mod
- How to calculate and review feature importance fr
- How to calculate and review permutation feature

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

< How to Develop Multi-Output Regression Models with Python

Gradient Boosting with Scikit-Learn, XGBoost, LightGBM, and CatBoost >

39 Responses to *How to Calculate Feature Importance With Python*

Martin March 30, 2020 at 6:35 pm # REPLY

This tutorial lacks the most important thing – comparison between feature importance and permutation importance. Which to choose and why?

For interested: https://explained.ai/rf-importance/

Best method to compare feature importance in Generalized Linear Models (Linear Regression, Logistic Regression etc.) is multiplying feature coefficients with standard devation of variable. It gives you standarized betas, which aren't affected by variable's scale measure. Thanks to that, they are comparable.

Scaling or standarizing variables works only if you have ONLY numeric data, which in practice… never happens.

Jason Brownlee March 31, 2020 at 7:59 am #

Comparison requires a context, e.g. a spe
of models.

Oliver Tomic March 30, 2020 at 7:54 pm #

Hi Jason!

Thanks for the nice coding examples and explanation. A little comment though, regarding the Random Forest feature importances: would it be worth mentioning that the feature importance using

importance = model.feature_importances_

could potentially provide importances that are biased toward continuous features and high-cardinality categorical features?

best wishes
Oliver

Jason Brownlee March 31, 2020 at 8:06 am # REPLY

It may be, what makes you say that?

Oliver Tomic March 31, 2020 at 5:48 pm # REPLY

I came across this post a couple of years ago when it got published which discusses how you have to be careful interpreting feature importances from Random Forrest in general. This was exemplified using scikit learn and some other package in R.

https://explained.ai/rf-importance/index.html

This is the same that Martin mentioned above.

best wishes
Oliver

Jason Brownlee April 1, 2020 at 5:49 am # REPLY

Thanks for sharing.

Aventinus March 30, 2020 at 11:22 pm #

Thank you, Jason, that was very informative.

As a newbie in data science I a question:

Is the concept of Feature Importance applicable to all r
What about BERT? I'm thinking that, intuitively, a simila
used, but when searching online I find that the answer
here.

REPLY

Jason Brownlee March 31, 2020 at 8:12 am #

Yes, but different methods are used.

Here, we are focused on tabular data.

REPLY

Alex March 31, 2020 at 1:04 am #

Hi, I am a freshman and I am wondering that with the development of deep learning that could find feature automatically, are the feature engineering that help construct feature manually and efficently going to be out of date? If not, where can we use feature engineering better than deep learning?

REPLY

Jason Brownlee March 31, 2020 at 8:13 am #

It performs feature extraction automatically.

Even so, such models may or may not perform better than other methods.

REPLY

Alex April 2, 2020 at 6:58 pm #

Thank you.

REPLY

Jason Brownlee April 3, 2020 at 6:51 am #

You're welcome.

REPLY

Fotis April 1, 2020 at 7:28 am #

Hi, I am freshman too. I would like to ask if the

Importance for Classification" using deep NN with Kera

Jason Brownlee April 1, 2020 at 8:10 am #

I don't see why not. Use the Keras wrappe

Ruud Goorden April 3, 2020 at 6:10 am #

REPLY

Hi. Just a little addition to your review. Beware of feature importance in RFs using standard feature importance metrics. See: https://explained.ai/rf-importance/
Keep up the good work!

Jason Brownlee April 3, 2020 at 6:59 am #

REPLY

Thanks for sharing.

Bill April 3, 2020 at 7:10 am #

REPLY

Hi Jason,

Any plans please to post some practical stuff on Knowledge Graph (Embedding)?

Thanks,Bill

Jason Brownlee April 3, 2020 at 7:56 am #

REPLY

Thanks for the suggestion Bill!

Ricardo April 5, 2020 at 10:31 pm #

REPLY

Your tutorials are so interesting.

Jason Brownlee April 6, 2020 at 6:05 am #

REPLY

Thanks.

Van-Hau Nguyen April 6, 2020 at 1:57 pm #

Hi Jason,

thank you very much for your post. It is very interesting
May I conclude that each method (Linear, Logistic, Ra
to Calculate Feature Importance?

Thank you.

Jason Brownlee April 7, 2020 at 5:36 am #

REPLY

Thanks.

Yes, we can get many different views on what is important.

Mayank April 11, 2020 at 9:01 pm #

REPLY

Hey Jason!!

Does this method works for the data having both categorical and continuous features? or we have to separate those features and then compute feature importance which i think wold not be good practice!.

and off topic question, can we apply P.C.A to categorical features if not then is there any equivalent method for categorical feature?

Jason Brownlee April 12, 2020 at 6:20 am #

REPLY

I believe so.

No. PCA is for numeric data.

Mayank April 12, 2020 at 1:56 pm #

REPLY

And L.D.A is for categorical values??

Jason Brownlee April 13, 2020 at 6

REPLY

LDA – linear discriminant analysis

Dina April 13, 2020 at 1:04 pm #

Hi Jason, I learnt a lot from your website abou
on how to know feature importance that use keras mod

Jason Brownlee April 13, 2020 at 1:52 pm # REPLY

Thanks.

Try: Permutation Feature Importance

Sam April 18, 2020 at 3:05 am # REPLY

Hi Jason,

I'm a Data Analytics grad student from Colorado and your website has been a great resource for my learning!

I have a question about the order in which one would do feature selection in the machine learning process. My dataset is heavily imbalanced (95%/5%) and has many NaN's that require imputation. A professor also recommended doing PCA along with feature selection. Where would you recommend placing feature selection? My initial plan was imputation -> feature selection -> SMOTE -> scaling -> PCA.

For some more context, the data is 1.8 million rows by 65 columns. The target variable is binary and the columns are mostly numeric with some categorical being one hot encoded.

Appreciate any wisdom you can pass along!

Jason Brownlee April 18, 2020 at 6:09 am # REPLY

Thanks, I'm happy to hear that.

Experiment to discover what works best.

I would do PCA or feature selection, not both. I would probably scale, sample then select. But also try scale, select, and sample.

Mbonu Chinedu April 25, 2020 at 7:09 am # REPLY

Thanks Jason for this information.

Jason Brownlee April 25, 2020 at 8:22 am #

You're welcome!

Deeksha April 29, 2020 at 11:17 pm #

Hi Jason,

I am running Decision tree regressor to identify the most important predictor. The output I got is in the same format as given. However I am not being able to understand what is meant by "Feature 1" and what is the significance of the number given.

I ran the Random forest regressor as well but not being able to compare the result due to unavailability of labelS. Please do provide the Python code to map appropriate fields and Plot.

Thanks

Jason Brownlee April 30, 2020 at 6:44 am #

REPLY

If you have a list of string names for each column, then the feature index will be the same as the column name index.

Does that help?

Swapnil Bendale May 3, 2020 at 3:47 pm #

REPLY

Sir,

How about using SelectKbest from sklearn to identify the best features???
How does it differ in calculations from the above method?

Thankin advance

Jason Brownlee May 3, 2020 at 5:10 pm #

REPLY

Yes, it allows you to use feature importance as a feature selection method.

Alex May 8, 2020 at 7:36 am #

Hi Jason,

Great post an nice coding examples. I am quite new to
own dataset and fitted a simple decision tree (classifier
when checking the feature importance. They were all 0
even possible?

Best
Alex

Jason Brownlee May 8, 2020 at 8:02 am #

REPLY

Thanks.

65% is low, near random. Perhaps the feature importance does not provide insight on your dataset. It is not absolute importance, more of a suggestion.

Alex May 8, 2020 at 9:02 am #

REPLY

ok thanks, and yes it's really almost random. But still, I would have expected even some very small numbers around 0.01 or so because all features being exactly 0.0 … anyway, will check and use your great blog and comments for further education . thanks

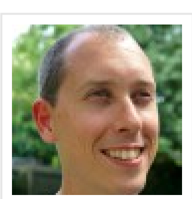
Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT



Welcome!

My name is *Jason Brownlee* PhD, and I **hel**

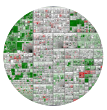
Read more

Never miss a tutorial:

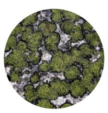
Picked for you:


Why One-Hot Encode Data in Machine Learning?


An Introduction to Feature Selection


How to Prepare Data For Machine Learning


How to Handle Missing Data with Python


Discover Feature Engineering, How to Engineer Features and How to Get Good at It

Loving the Tutorials?

The EBook Catalog is where I keep the ***Really Good*** stuff.

SEE WHAT'S INSIDE